Tutorial - 1

A-1 → Asymptotic Notations is used to describe the running time of an Algorithm. This shows the order of growth of function.

Different types of Asymptotic Notations :-

1. Big-O Notation → It represents asymptotic upper bound of the function.

$$f(n) = O(g(n)) \text{ if } f(n) \leq c.g(n)$$

2. Big-Ω Notation → It represents lower bound of Algorithm.

$$f(n) = \Omega(g(n)) \text{ if } f(n) \geq c.g(n)$$

3. Theta Θ Notation → It represents upper and lower bound of a function.

$$f(n) = \Theta(g(n)) \text{ if } C_1 g(n) \leq f(n) \leq C_2 g(n)$$

A-2 →
```
for (i = 1 to n)
{
    i *= 2
}
```

$i = 1$
$i = 2$
$i = 4$
$\vdots$
$i = n$

$$a_n = a \cdot r^{n-1}$$
$$n = 1 \times 2^{K-1}$$
$$\log n = (K-1) \log_2 2 \Rightarrow K = \log n + 1 = O(\log n)$$

Ans-3 → $T(n) = 3T(n-1) \quad n>0$

$T(0) = 1$

$T(n) = 3T(n-1)$

$T(n-1) = 3T(n-2)$

$T(n) = 3(3T(n-2))$

$T(n-2) = 3\,T(n-3)$

$T(n) = 3(3(3T(n-3)))$

$T(n) = 3^k\,T(n-k)$

$n-k=0$

$n=k$

$T(n) = 3^k\,T(n-n) = 3^n\,T(0) = 3^n \cdot 1$

$T(n) = 3^n = O(3^n)$

Ans-4 → $T(n) = 2T(n-1) - 1$

$T(0) = 1$

$T(n) = 2T(n-1) - 1$

$T(n-1) = 2T(n-2) - 1$

$T(n) = 2(2T(n-2)-1) - 1$

$T(n-2) = 2T(n-3) - 1$

$T(n) = 2(2(2T(n-3)-1)-1)-1$

$$T(n) = 2^3 T(n-3) - 2^2 - 2^1 - 2^0$$

$$T(n) = 2^k T(n-k) - 2^{k-1} - 2^{k-2} - 2^{k-3} - \dots - 2^0$$

$$n - k = 0 \Rightarrow n = k$$

$$T(n) = 2^n - 2^{n-1} - 2^{n-2} \dots \dots - 2^0$$

$$T(n) = 2^n - \sum_{i=0}^{n-1} 2^i$$

$$= 1$$

$$\Rightarrow O(1)$$

Ans-5 →

```
int i=1, s=1
while (s<=n)          ——— ?
{   i++;              ——— O(1)
    s+=i;             ——— O(1)
    print("#");       — O(1)
}
```

$i = 1 \quad s = 1$

$i = 2 \quad s = 1 + 2$

$i = 3 \quad s = 1 + 2 + 3$

$i = 4 \quad s = 1 + 2 + 3 + 4$

$\vdots \quad \vdots$

$$s > n$$

$$1 + 2 + 3 + 4 + \dots \dots k > n \qquad \frac{k(k-1)}{2} > n$$

$K^2 > n$, $K > \sqrt{n}$ $\Rightarrow O(\sqrt{n})$

A-6 → for (i = 1; i*i <= n; i++)

Count++;

$i = 1$	$i \times i = 1$	$i \times i > n$
$i = 2$	$i \times i = 4$	$K \times K > n$
$i = 3$	$i \times i = 9$	$K^2 > n$
$i = 4$	$i \times i = 16$	$K > \sqrt{n}$
⋮	⋮	$O(\sqrt{n})$
	$i \times i > n$	

A-7

```
int i, j, k.
for (i = n/2 ; i <= n ; i++)
    for (j = 1 , j <= n ; j = j * 2)
        for ( k = 1 ; k <= n ; k = k * 2 )
```

i	j	k
$n/2$	$1, 2, 4, 8, \dots n = \log n$	$1, 2, 4, 8 \dots n = \log n \times \log n$
$n/2+1$	⋮	⋮
$n/2+2$	⋮	⋮
⋮		
n		

$\frac{n}{2}$

$\Rightarrow O(n)$ $O(\log n)$ $O(\log n) \Rightarrow O(n \log^2 n)$

Ans-8 → $T(n) = T(n-3) + n^2$

$T(1) = 1$

$T(1) = 1$

$T(4) = T(4-3) + 4^2$

$= T(1) + 4^2 = 1^2 + 4^2$

$T(7) = T(7-3) + 7^2$

$= 1^2 + 4^2 + 7^2$

$T(10) = T(10-3) + 10^2$

$= 1^2 + 4^2 + 7^2 + 10^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 \ldots n^2 = \frac{n(n+1)(2n+1)}{6}$

$= O(n^3)$ also for terms like $T(2)$, $T(3)$.

So, $T(n) = O(n^3)$

A-9 →

```
void funct (int n)
{ for(int i=1 to n)
  { for(j=1; j<=n; j+i)..
    { printf("*");
    }
  }
}
```

i=1 j=1 to n

i=2 j=1 to n

i=3 j=1 to n

i=4

⋮

So, for upto n it will take n^2.

So, $T(n) = O(n^2)$

A-10 → $f(n) = n^k$, $f_2(n) = c^n$

Asymptotic relationship between f_1 and f_2 $k >= 1$

is Big O i.e. $f_1(n) = O(f_2(n)) = O(c^n)$ $c > 1$

is $n^k \leq C_1 * c^n$ [C_1 is some constant]